A Loose and Sketchy Approach in a Mediated Reality Environment

Michael Haller*, Florian Landerl†
Media Technology and Design
Upper Austria University of Applied Sciences

Mark Billinghurst‡
HITLabNZ
University of Canterbury

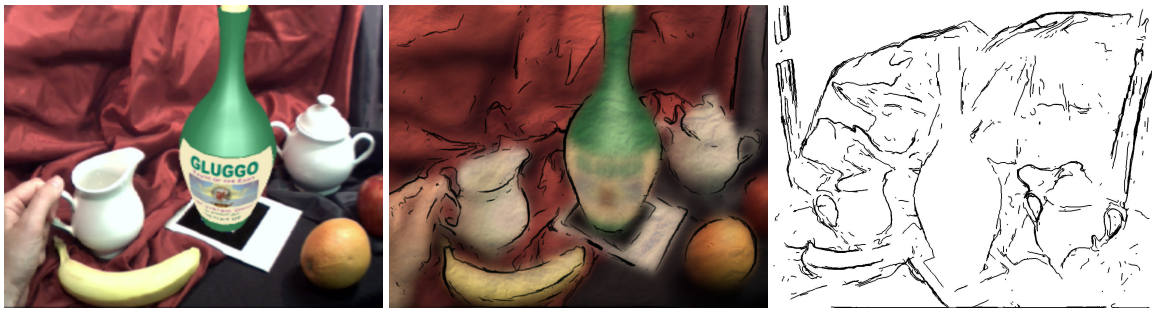


Figure 1: A traditional rendered AR scenario (a) and the NPR rendered scenario (b), (c).

Abstract

In this paper, we present sketchy-ar-us, a modified, real-time version of the Loose and Sketchy algorithm used to render graphics in an AR environment. The primary challenge was to modify the original algorithm to produce a NPR effect at interactive frame rate. Our algorithm renders moderately complex scenes at multiple frames per second. Equipped with a handheld visor, visitors can see the real environment overlaid with virtual objects with both the real and virtual content rendered in a non-photorealistic style.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation— [I.3.5]: Non Photorealistic Rendering—Hardware Accelerated Rendering H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities

Keywords: Mediated Reality, Non-photorealistic rendering, Loose and Sketchy

1 Introduction

In recent years, non-photorealistic rendering (NPR) has become a popular research topic in the area of computer graphics. Augmented Reality applications are those in which computer graphics are overlaid in real time on views of the real world. In Augmented Reality (AR) installations, there are two alternatives for rendering the superimposed 3D content: either using photorealistic rendering techniques with the goal of seamlessly integrating the augmented content into the existing real environment, or using an NPR style to enhance the augmented content. In the last decade AR research has mostly been focused on improving photorealistic rendering, including consistent illumination, integrating shadows and enhancing shading by BRDFs etc. [Agusanto et al. 2003; Bimber et al. 2003; Gibson et al. 2003; Haller et al. 2003; Naemura et al. 2002; Sugano et al. 2003]. However there may be other ways to make AR installations more visually appealing and fun. Recently there has been new research trying to making AR installations more stylistically believable, and/or more enjoyable, than photorealistic.

Ferwerda [Ferwerda 2003] distinguished three different varieties of realism:

- physical realism, where the virtual objects provide the same visual simulation as the real scene.
- photorealism, where the image produces the same visual response as the scene, and
- functional realism, in which the image provides the same visual information as the scene.

In this paper, we focus on aspects of photorealism (non-photorealism) in AR interfaces and leave addressing physical realism (non-realism) and functional realism (non-realism) for later work. There are a lot of good reasons to improve the realism of augmented reality imagery [Haller 2004]. Currently it is challenging to create a seamless (not discontinuous) coherence between the real and the virtual world: for example, light sources have to influence the augmented content in the same way as they affect real objects, including casting virtual shadows to match real shadows. Gogolin predicts that sooner or later, photorealism research will start developing very new rendering techniques [Gogolin 2004]. Durand [Durand 2002] demonstrates that the border between photorealism and non-photorealism can be fuzzy and the idea of realism itself can become very complex. The virtual world has to be interpreted more *convincingly* rather than *realistically rendered*. In fact, it should be a *believable* world, where the augmented objects should be expressive, clear, and look aesthetically perfect.

The goal of this work is to achieve a more stylistic and artistic AR visualization (cf. figure 2).

The main contributions of this paper are twofold:

*e-mail:haller@fh-hagenberg.at
†e-mail:florian.landerl@fh-hagenberg.at
‡e-mail:mark.billinghurst@hitlabnz.org

Figure 2: sketchy-ar-us in use: the embedded snapshot shows the view of what the users would see through the handheld visor.

Implementation of a real-time version of the Loose and Sketchy algorithm using programmable GPU hardware. Curtis' original algorithm was primarily designed for offline rendering, where the performance played a secondary role. Thus, it took 10-60 seconds to render each frame [Curtis 1999].

Combination with an AR environment: Both the real and the virtual objects are rendered in the same sketchy style. Due to the blurred image and the sketchy silhouette style, users cannot differ between the real and the virtual, augmented objects. Consequently, they get a better immersive experience and a more enjoyable stylistic view.

After an overview of related work, described in section 2, we demonstrate our approach, the modified Loose and Sketchy algorithm, including the creation of the silhouettes, the blurring of the image, its composition, and the combination with the AR setup. Performance tests and user feedback comments are discussed in section 5. Finally, we conclude the paper with directions for future work.

2 Related Work

Our work is based on elements of Mediated Reality, as introduced by Steve Mann [Mann and Fung 2001; Mann 1994]. In contrast to Augmented Reality, in a Mediated Reality interface virtual content is not just added to the real environment, but modified by a visual filter. Grasset et al. [Grasset et al. 2003] demonstrate an application where they allow a user to paint virtually onto a real environment. In this paper, we present sketchy-ar-us (cf. figure 1), a modified, real-time version of the Loose and Sketchy algorithm presented by Curtis [Curtis 1999] used to render an AR environment using ARToolKit [Kato et al. 1999].

Sketchy-ar-us presents a non photorealistic view of the AR scene using non-photorealistic rendering (NPR). There are several reasons why NPR images may be better for some applications: the pictures are easy to understand, they are easy to display, and they do not need a huge amount of data [Raskar et al. 2004]. A hand-drawn sketch can often communicate complex coherences in a better way than photorealistic pictures. Sketchy (un-completed) drawings need users to mentally complete the picture and add missing details. McCloud describes in [McCloud 1994] that scenarios can be expressed easier by using a simple, comic style.

A general overview of different NPR algorithms are given by [Gooch and Gooch 2001; Strothotte and Schlechtweg 2002]. Many algorithms have been developed in the last decade that create images which resemble art made by humans [Gooch et al. 2002; Hertzmann and Perlin 2000]. Different art techniques and styles can be simulated, such as pen and ink [Salisbury et al. 1994], hatching [Praun et al. 2001], water color [Curtis et al. 1997] and impressionism [Haeberli 1990; Meier 1996].

Although NPR techniques are becoming more common, there has been little research conduced into combining AR content with novel rendering techniques. In [Haller and Sperl 2004] we presented a non-photorealistic renderer in an AR environment, where only the virtual objects have been rendered in a painterly style. Fischer et al. [Fischer et al. 2005] postulate a cartoon-like AR environment, where both the virtual and the real objects are rendered in the same style. Their approach is based on a bilateral image filtering for the color segmentation and a Canny-edge-detector for the silhouette generation. Their work motivated us to combine the Loose and Sketchy algorithm with an AR setup to achieve a more stylistic and artistic environment.

Another example of how the AR content can be enriched by NPR objects is presented by Collomosse et al. [Collomosse et al. 2003a; Collomosse et al. 2003b], who postulate that an abstract illustration of motion also makes sense for real movie sequences. In their work, they enhance the motion in movies by adding motion lines or deforming real objects so that they seem to be moving quickly. Their movies are impressive and convincing, although their techniques require extensive pre-processing.

Our Loose and Sketchy algorithm consists of three different steps: finding the silhouette by using reference images, blurring the image to achieve a more fuzzy image, and adding paper texture to produce a more stylistic image. A lot of different algorithms have been published for the generation of silhouettes [Saito and Takahashi 1990; Card and Mitchell 2002; Raskar and Cohen 1999; Northrup and Markosian 2000]. Kowalski et al. [Kowalski et al. 1999] demonstrate a novel silhouette rendering technique to achieve an art-based rendering of fur, grass, and trees. However, less attention has been given to coherent stylized silhouettes (cf. [Masuch et al. 1997; Kalnins et al. 2003]).

In contrast to Fischer et al. [Fischer et al. 2005] our algorithm uses the basic concepts proposed by Curtis. In our case, both the virtual content (3d scenario) and the real scenario (the video input) uses a "different" rendering mechanism. Consequently, for both images (the augmented, virtual content and the real scene) we used different input to achieve the best NPR results. So, for example, we used the depth-information to create silhouettes of the virtual 3d objects. In addition, 3d objects closer to the camera viewpoint could be drawn with thicker silhouette lines than objects that were far away. Fischer's algorithm is mainly based on edge-detection and it renders the silhouette edge in a normal way. In contrast, our algorithm is based on a particle system, which allows more flexibility and results in a more stylistic image.

3 Real-time Loose and Sketchy approach

The Loose and Sketchy technique of Curtis produces images that appear to be drawn by hand. It automatically draws the visible silhouette edges of a 3D model using image processing and a stochastic, physically-based particle system.

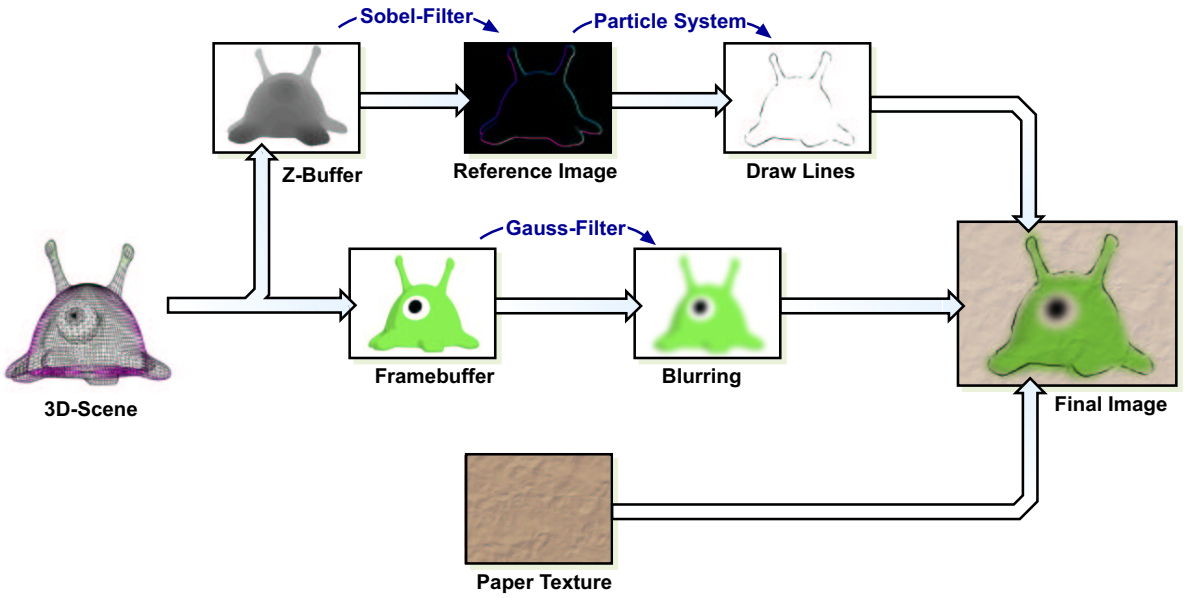


Figure 3: The pipeline of the real-time Loose and Sketchy algorithm.

Due to its complexity, the original Loose and Sketchy method could not be used for real-time applications. To apply the same style of rendering in real-time, we modified the technique of Curtis by extensively using modern 3D hardware—particularly the programmable graphics pipeline. The real-time method is still based on the same principles as laid out in [Curtis 1999], albeit with minor simplifications.

In the following sections the steps involved in creating a real-time Loose and Sketchy image are discussed. The graphics pipeline of the described system is shown in figure 3.

3.1 Preprocessing

The Loose and Sketchy algorithms rely heavily on image processing, both for edge detection as well as the blurring of the scene. The edge detection filter operates on the depth-buffer and the blur filter manipulates the color-buffer. To perform these filters successfully, the appropriate buffers have to be prepared first.

Direct rendering to textures using *PBuffers* [Wynn 2002] allows for rapid generaton of the necessary data. Once defined as a render target, the rendering process draws directly into the appropriate color- and depth-buffer textures of the *PBuffer*—without taking a detour over the standard on-board frame- and depth-buffer. Afterwards these textures can be used just like normal OpenGL textures.

3.2 Generating the reference image

The reference image holds two different kinds of data important for drawing the strokes. Firstly, it contains the silhouette information of the scene and, secondly, a ”force field” which is used to place the strokes along the silhouettes. By applying the Sobel edge detection filter on the depth-buffer texture, the necessary data can be found for constructing the reference image.

In contrast to Fischer et al. [Fischer et al. 2005], we do not use the Canny filter, because we needed to create ”force field” vectors to determine the movement of the particles.

The Sobel operator performs a 2D spatial gradient measurement on an image and then emphasizes regions of high spatial gradient that correspond to edges. Since discontinuities in neighboring depth-buffer values occur mostly at the 3D-objects’ contours, only silhouette edges and a few boundary edges will be detected—but this is exactly what is needed. In [Saito and Takahashi 1990], Saito and Takahashi introduced how to find boundary edges by taking into account discontinuities in neighboring surface normal values.

The Sobel edge detection filter consists of a pair of 3×3 convolution masks shown in equation 1, where the second matrix is simply the transposition of the first one.

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{1}$$

The masks are applied separately to the depth-buffer, to produce separate measurements of the gradient component in each orientation. These components are then combined to form a 2D vector.

The absolute magnitude of this gradient vector is used to determine if an edge has been found at the current pixel position. In order to assert an edge, the absolute magnitude of the gradient vector must exceed a defined threshold value. The lower the threshold, the more edges that will be found, since a lower magnitude is sufficient for ”edge-qualification”—a higher threshold results in fewer detected edges. The ”force field” is obtained by calculating unit vectors perpendicular to the gradient vector.

The reference image’s creation is completely performed inside a fragment shader and the results of the fragment program are written to a texture. Therefore, the silhouette information as well as the ”force field” data has to be encoded as pixel values. The pixel’s blue component contains a value which determines whether an edge has been found or not. If no edge has been detected this value is set to 0. However, if an edge is present, the depth-buffer’s data of the

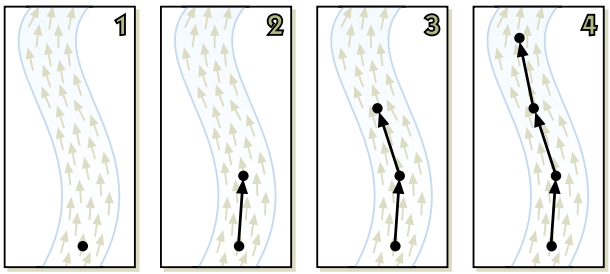


Figure 4: Using the ”force field”, new particles are created along the silhouette.

current pixel location is stored instead. This information can be useful later on when drawing the lines. The red and green values of the pixel hold the x- and y-components of the ”force field” vector respectively.

3.3 Drawing the outlines

The reference image shows where the objects' silhouettes are to be found. The final image consists of individual brush strokes that are drawn along these silhouettes. After calculating the reference image using a fragment shader, it is stored as a texture. Since the reference image has to be accessible when drawing the strokes, the contents of this texture have to be read back into main memory. Unfortunately, this ”memory read-back” is a very demanding process, and is to be held as the scapegoat for being the major bottleneck in the whole application.

For rendering the brush strokes on the screen we implemented a special particle system. A single stroke consists of multiple particles, which determine the stroke's position and course. For each frame a predefined number of particles are consecutively emitted, and placed randomly on the screen. By looking up the reference image, we assert if the particle's position is located at a silhouette edge. If this is not the case the particle gets deleted immediately.

If the particle is located on a silhouette edge, it is defined as the starting point of a new brush stroke. Moving some steps from this starting point along the direction of the previously created ”force field”, the next point of the stroke is derived. In a similar manner, now starting from the second point, a further point of the stroke is determined. This process is reiterated until a point departs from the silhouette edge or the desired stroke length is exceeded (cf. figure 4).

After this procedure, the particles which make up a single stroke are exactly specified. Apart from the position, a particle stores additional parameters which can be used to modify the visual appearance of the stroke. For example, by taking information of the original depth-buffer into account, more distant objects can be represented by thinner brush strokes to give hints on the depth conditions of a scene.

The process of generating a single brush stroke is repeated until the desired number of strokes is reached. As soon as the characteristics for all strokes are defined, they can finally be drawn onto the screen.

We implemented three different methods to render the strokes:

1. The strokes are rendered using textured polygons (cf. figure 5 (a)). This method displays changes in the visual appearance of a stroke with a smooth transition using modern graphics hardware. For example, by increasing the transparency along the stroke's length, a subtle fade-out for the brush strokes can be achieved. Furthermore, the line thickness can be precisely set by appropriately adjusting the polygons' size. Another major advantage is the possibility of rendering the polygons with textures attached to them. By the use of brush textures, a multitude of different drawing styles can be imitated, as can be seen in table 2. Thus, this approach represents the most artistically versatile style. The silhouettes were created by using normal quad strips and attaching brush strokes. To guarantee that this does not result in ”stretching artifacts”, the brush stroke textures have to be chosen carefully (cf. table 2).

2. The brush strokes are rendered using the OpenGL `GL_LINE_STRIP` command (cf. figure 5 (b)). This solution is not well suited for the task of drawing brush strokes. The biggest drawback of this method is that smooth transitions of transparency, stroke width or color, cannot be achieved. Moreover, the lines tend to be aliased and therefore they produce artifacts that interfere with the perception of a drawn style.

3. An interesting option is to use the particles in a more conventional manner. Instead of generating brush strokes, each particle stands on its own and a bitmap is rendered at its position (cf. figure 5 (c)). A particle stays alive for a certain period of time and moves along the silhouette using the reference image. The particle is destroyed after its time expires—not when leaving the silhouette. A problem with this approach appears when the objects in the scene move rather fast, causing their silhouette to change dramatically per frame. Since the particles stay on the screen for some time and travel only a small distance, the result lags behind the current proper silhouettes.

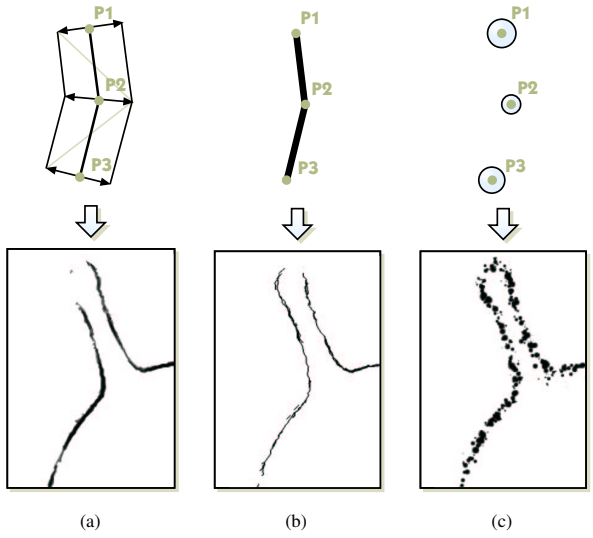


Figure 5: The various rendering methods produce distinctively different results.

The reference image is computed for every single frame to properly reflect changes in the 3D scene. Because of this, the brush strokes also have to be redrawn in every frame. Since the strokes are created randomly, they are different every time and a frame-to-frame coherence between the brush strokes is not guaranteed.

3.4 Blurring the background

The techniques presented so far are capable of rendering 3D scenes as pure outline drawings. In order to create the final image the

bodies of the sketched objects are filled with color. Color provides a better understanding and allocation of the displayed scene.

The color information obtained by a standard rendering pass serves as a basis for the color fill. As discussed in 3.1 a texture was prepared containing the color-buffer of the rendered scene. By blurring this texture, the colors appear to fill the strokes completely despite the fact that the strokes do not coincide with the edges of the objects. This works because our brains process color and edge information separately [Curtis 1999]. Blurring the color texture removes the high-frequency information that would otherwise cause an impression of a double edge.

The blurring is accomplished in hardware using a two-dimensional image-processing filter as described in [Fernando 2004, Cha. 21]. First, the color-buffer texture is blurred in one axis to produce a temporary image. This image is then blurred in the other axis to produce the final blur. This procedure is demonstrated in figure 6.

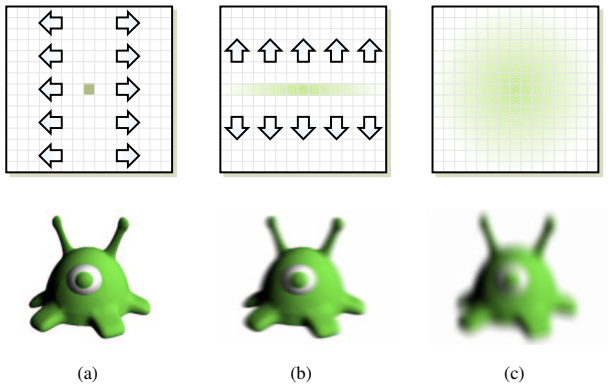


Figure 6: The Gaussian blur is implemented as a two-step operation.

By implementing the blur as a two-step operation, the processing time is reduced dramatically. The time required to perform a blur is dependent on the proportion of the blur's area. The larger the blur the more surrounding pixels that have to be taken into account. In a single pass, the area is proportional to the blur diameter squared (d^2). This would make large-area blurs impractical for real-time usage. By blurring the image in a two-step operation the cost is reduced from d^2 to $2 \cdot d$, which can easily be handled by modern graphic cards.

The source code of a Gaussian blur using a two-step approach is available from the *Scotopic Vision Demo* from nVIDIA's SDK[1]. We basically used the same code for creating the blur effect in the real-time Loose and Sketchy renderer.

3.5 Putting it together

First, the blurred color texture is rendered onto the screen. Afterwards, to enhance the "handmade" illusion, we add a paper texture to the rendering. Finally, the brush strokes are drawn on top of the composed image. All the steps involved in creating a Loose and Sketchy rendering are presented below (and shown visually in figure 3):

[1] http://developer.nvidia.com

```
for each frame of animation
  render scene to PBuffer
  generate reference image from depth-buffer
  update particle system with reference image
  blur color-buffer texture

  render blurred color texture to framebuffer
  multiply framebuffer by paper texture
  for each particle (that starts a stroke)
    add brush stroke to framebuffer
  end for
end for
```

4 Integrating real-time Loose and Sketchy into the AR environment

In a video see-through AR environment, a video stream from a camera is acquired and rendered as a background image for the virtual 3D objects. In order to achieve a seamless combination of both the real and the virtual objects, and thereby an immersive impression of a *believable* world, everything has to be drawn in the same visual style.

To be able to apply the brush strokes that are used for rendering the 3D objects' silhouettes, to the background image as well, a reference image for the real environment has to be generated. By applying the same Sobel edge detection filter, albeit with higher threshold settings to the red channel of the background image, we are able to create decent silhouette information for the real environment. In real-life scenarios, a certain amount of noise cannot be avoided (at least not without prior processing), and we have to compensate this flaw by setting a higher threshold. Since the contents of the depth-buffer are strictly computer generated, there is no need to worry about noise in the image and we generally get away with a lower threshold.

When creating the 3D objects' reference image, the silhouette information of the real environment is passed to the shader program. The fragment shader calculates the objects' silhouettes and also merges both reference images into one which is subsequently used to place the brush strokes onto the screen. By calculating the silhouettes of the real and virtual images separately, and subsequently merging them, we are able to control the parameters of the edge detection processes individually to achieve better results.

The blurring of both the real and the virtual objects is achieved by rendering the 3D objects on top of the acquired video frame, and processing the two-step blur on that image. After the paper texture is blended, we draw the brush strokes to get the final Loose and Sketchy result in an AR environment.

5 Results and Discussions

Figure 8 (a) shows a traditional AR rendered scene of an augmented (not realistic rendered) bottle. Figures (b) to (d) represent different stages of the Loose and Sketchy algorithm. The installation has been presented at the local city art gallery, where about 200 people had the possibility to give their feedback. It was interesting to see that people really loved to watch themselves in a different style (cf. figures (d), (e), and (f)). They liked the stylistically rendered scene much more than the Gouraud shaded still life. They were amazed by the fact that they could not distinguish between the real and the virtual objects while using a handheld visor.

After blurring the scene, the augmented and the real objects were difficult to distinguish. Some of the visitors criticized the fact that the whole scene was too blurred; they were thinking that the camera's lens was incorrectly calibrated. However, the combination with the paper texture for the background gave the scene the impression of a painted, sketched image. Finally, by using a sketched border, people really had the impression of watching a sketched AR environment.

All scenes were rendered on a 2.8 GHz PC with 1 GB of memory using an nVIDIA Geforce FX 6800 with 256 MB. The performance does not depend on the complexity of the rendered model, thus in our case we achieved 16.87 fps. A more detailed overview of the frames-per-seconds is given in table 1, which shows different scenarios rendered in the blurred style, adding the sketchy silhouette, and adding the background paper. In the two testing environments, we used a model with 894 polygons for the first scene (bottle geometry) and 10,182 polygons for the second scene (Van Gogh's bedroom geometry) respectively. Notice that the performance was measured in combination with the ARToolKit for marker detection and using an ADS webcam delivering a resolution of 640x480 pixels at 30 fps. One of the biggest bottlenecks, was the ARToolKit marker detection library, which runs on the CPU rather than the GPU.

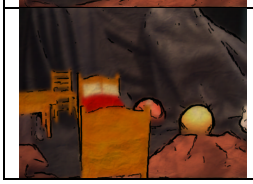
Scenario	Shading	fps
	Add blurring Add edges Add paper	18.51 16.95 16.87
	Add blurring Add edges Add paper	18.40 16.80 16.54

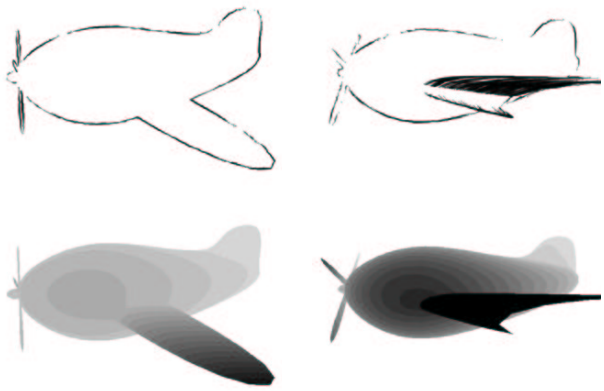
Table 1: The performance of sketchy-ar-us in two different scenarios.

In contrast to [Fischer et al. 2005], our approach uses a completely different method for rendering the scene in a non-photorealistic style. In our case, the silhouette strokes are placed randomly, which itself results in a more "dynamic" image. This is the desired effect, since it animates the outcoming image and gives the scene a life of its own.

Thanks to the blurring effect, we didn't have to care about a one-to-one matching of the real scenario properties with those of the virtual content (i.e. guaranteeing the same lighting parameters and light positions as in the real environment). The missing shadows, the wrong light position and the corresponding wrong shading (because of the fixed OpenGL light position) was never criticized by the visitors.

In our example, the virtual and the real images have been sent individually to the Loose and Sketchy pipeline. Thus, the parameters for the silhouettes (e.g. the edge detection) could be controlled individually to achieve better results. Table 2 shows some brush strokes used for the silhouette and the corresponding close-up of the stylistic rendered image.

One problem with our rendering approach occurs with the usage of the depth-buffer for the silhouette detection. Long drawn-out surfaces pointing towards the user can cause unwanted silhouette detection inside the surface. This is because the depth information inside the shape changes extremely, which results in depth layers, interpreted as silhouettes (cf. figure 7).



(a) (b)

Figure 7: Unwanted silhouettes might occur in long drawn-out surfaces.

6 Future Work and Conclusion

In this paper, we have presented a real-time version of Curtis' algorithm for rendering AR images in a Loose and Sketchy style (cf. [Curtis 1999]). By using modern graphics hardware and high-level shader languages, the process is fast enough to be applied in real-time AR environments. NPR in an augmented environment can become more and more important, especially for artistic applications: by making images look less photorealistic, artists enable audiences to feel more immersed in a virtual world [McCloud 1994]. Our primary interest was to enable the use of real-time stylized rendering to create a more compelling and interesting environment.

One problem of our proposed algorithm was the lag of stylized silhouettes with a robust frame-to-frame coherence. The "flickering" effect, as it was proposed by Curtis, can become disturbing once the objects are moving fast. The particle position, which is calculated randomly within the silhouette path, has to be chosen more carefully from one frame to the other. Kalnins et al. proposed in [Kalnins et al. 2003] one possible solution for propagating the parametrization from strokes in one frame to strokes in the next. We would like to combine these presented ideas with our algorithm to achieve more compelling results.

Moreover, we want to adapt our algorithm not just on the whole frame and render both the real and the virtual content in the same style. We believe that in some cases it makes more sense to render just parts of the scenario in a non-photorealistic style to enhance regions of the scenario.

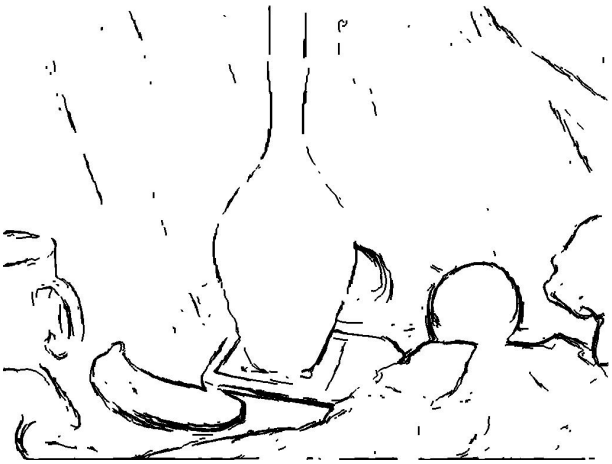
Finally, we want to start a formal usability study to get feedback what people expect from AR in combination with NPR. People have a wide different view regarding the benefits of a non-photorealistic rendered AR scenario and it would be intersting to find out in which sense it influences the visual perception.

(a) Gouraud rendered image.

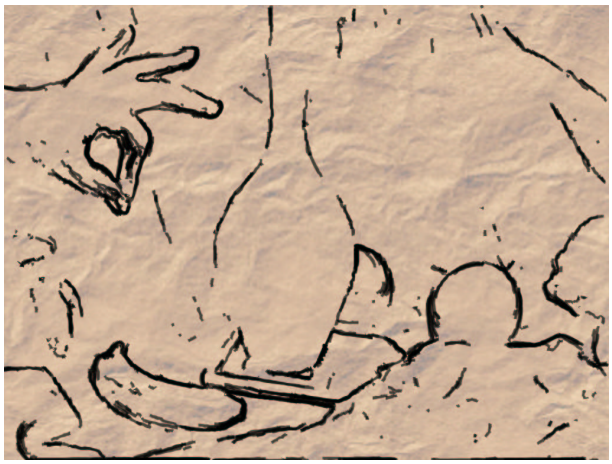
(b) Blurred colored image.

(c) Combination of color and paper texture.

(d) Adding particle rendered outlines.



(e) The still life without paper texture and shading color.



(f) Still life with paper texture but no shading color.

Figure 8: Different scenario results including some steps of the rendering process of sketchy-ar-us.

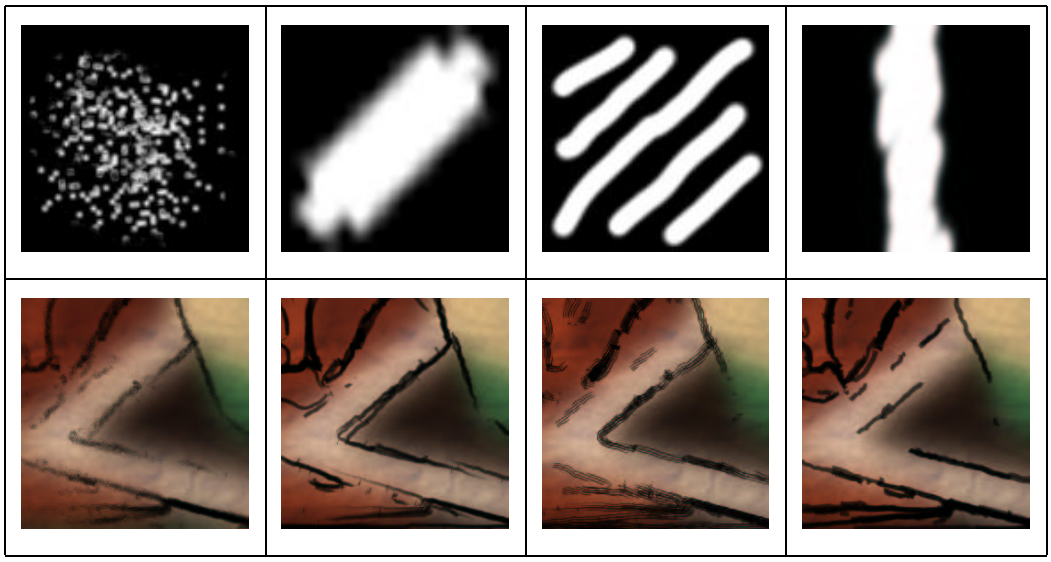


Table 2: Different brush textures for the strokes result in different appearances of the scenario.

References

AGUSANTO, K., LI, L., CHUANGUI, Z., AND SING, N. W. 2003. Photorealistic rendering for augmented reality using environment illumination. In *IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR 2003)*, IEEE Computer Society, 208–216.

BIMBER, O., GRUNDHOEFER, A., WETZSTEIN, G., AND KNOEDEL, S. 2003. Consistent illumination within optical see-through augmented environments. In *IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR 2003)*, IEEE Computer Society, 198–207.

CARD, D., AND MITCHELL, J. 2002. Non-Photorealistic Rendering with Pixel and Vertex Shaders. In *Direct3D ShaderX*, Wordware, W. Esngel, Ed.

COLLOMOSSE, J. P., ROWNTREE, D., AND HALL, P. M. 2003. Stroke surfaces: A spatio-temporal framework for temporally coherent non-photorealistic animations. Tech. Rep. 2003–01, University of Bath, U.K., Bath, June.

COLLOMOSSE, J. P., ROWNTREE, D., AND HALL, P. M. 2003. Video analysis for cartoon-like special effects. In *14th British Machine Vision Conference*, 749–758.

CURTIS, C. J., ANDERSON, S. E., SEIMS, J. E., FLEISCHER, K. W., AND SALESIN, D. H. 1997. Computer-generated watercolor. In *Proceedings of SIGGRAPH 97*, 421–430.

CURTIS, C. 1999. Non-photorealistic animation. In *Proc. ACM SIGGRAPH 1999*, ACM Press.

DURAND, F. 2002. An invitation to discuss computer depiction. In *Proceedings of the second international symposium on Non-photorealistic animation and rendering*, ACM Press, 111–124.

FERNANDO, R. 2004. *GPU Gems*. Addison-Wesley.

FERWERDA, J. 2003. Three varieties of realism in computer graphics. In *Proceedings SPIE Human Vision and Electronic Imaging '03*.

FISCHER, J., BARTZ, D., AND STRASSER, W. 2005. Stylized Augmented Reality for Improved Immersion. In *Proceedings of IEEE Virtual Reality (VR 2005)*.

GIBSON, S., COOK, J., HOWARD, T., AND HUBBOLD, R. 2003. Rapid shadow generation in real-world lighting environments. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, Eurographics Association, 219–229.

GOGOLIN, H. 2004. Wenn weniger mehr ist. *GEE Magazin 6* (August), 64–68.

GOOCH, A., AND GOOCH, B. 2001. *Non-Photorealistic Rendering*. AK Peters, Ltd., July 1. ISBN: 1568811330, 250 pages.

GOOCH, B., COOMBE, G., AND SHIRLEY, P. 2002. Artistic vision: painterly rendering using computer vision techniques. In *Proceedings of the second international symposium on Non-photorealistic animation and rendering*, ACM Press, 83–ff.

GOURAUD, H. 1971. Continuous shading of curved surfaces. In *IEEE Transactions on Computers*, 623–629.

GRASSET, R., GASCUEL, J., AND SCHMALSTIEG, D. 2003. Interactive Mediated Reality (poster). In *IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, 302.

HAEBERLI, P. E. 1990. Paint by numbers: Abstract image representations. In *Proceedings of SIGGRAPH 90*, 207–214.

HALLER, M., AND SPERL, D. 2004. Real-time painterly rendering for mr applications. In *Graphite, International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia*, to be published.

HALLER, M., DRAB, S., HARTMANN, W., AND ZAUNER, J. 2003. A real-time shadow approach for an augmented reality application using shadow volumes. In *ACM Symposium on Virtual Reality Software and Technology*.

HALLER, M. 2004. Photorealism or/and non-photorealism in augmented reality. In *VRCAI '04: Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*, ACM Press, 189–196.

HERTZMANN, A., AND PERLIN, K. 2000. Painterly rendering for video and interaction. In *Proceedings of the first international symposium on Non-photorealistic animation and rendering*, ACM Press, 7–12.

KALNINS, R. D., DAVIDSON, P. L., MARKOSIAN, L., AND FINKELSTEIN, A. 2003. Coherent stylized silhouettes. *ACM Trans. Graph. 22*, 3, 856–861.

KATO, H., BILLINGHURST, M., BLANDING, B., AND MAY, R. 1999. Artoolkit.

KOWALSKI, M. A., MARKOSIAN, L., NORTHRUP, J. D., BOURDEV, L., BARZEL, R., HOLDEN, L. S., AND HUGHES, J. F. 1999. Art-based rendering of fur, grass, and trees. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 433–438.

LANDERL, F. 2005. *Skizzenhaftes Rendering in Echtzeitanwendungen*. Master's thesis, Fachhochschule Hagenberg, Medientechnik und -design, Hagenberg, Austria.

MANN, S., AND FUNG, J. 2001. VideoOrbits on Eye Tap devices for deliberately Diminished Reality or altering the visual perception of rigid planar patches of a real world scene. In *International Symposium on Mixed Reality (ISMR)*.

MANN, S. 1994. Mediated reality. TR 260, M.I.T. Media Lab Perceptual Computing Section, Cambridge, Massachusetts, http://wearcam.org/mr.htm.

MASUCH, M., SCHLECHTWEG, S., AND SCHÖNWÄLDER, B. 1997. dali! - drawing animated lines! 87–95. ISBN 1-56555-111-7.

MCCLOUD, S. 1994. *Understanding Comics*. Perennial Currents.

MEIER, B. J. 1996. Painterly rendering for animation. In *Proceedings of SIGGRAPH 96*, 477–484.

NAEMURA, T., NITTA, T., MIMURA, A., AND HARASHIMA, H. 2002. Virtual Shadows - Enhanced Interaction in Mixed Reality Environment. In *IEEE Virtual Reality (VR'02)*.

NIENHAUS, M., AND DOELLNER, J. 2004. Blueprints: illustrating architecture and technical parts using hardware-accelerated non-photorealistic rendering. In *GI '04: Proceedings of the 2004 conference on Graphics interface*, Canadian Human-Computer Communications Society, 49–56.

NORTHRUP, J. D., AND MARKOSIAN, L. 2000. Artistic silhouettes: A hybrid approach. In *Proceedings of the First International Symposium on Non Photorealistic Animation and Rendering (NPAR) for Art and Entertainment*. To be held in Annecy, France.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-time hatching. In *Proceedings of SIGGRAPH 2001*, 581.

RASKAR, R., AND COHEN, M. 1999. Image precision silhouette edges. In *SI3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, ACM Press, 135–140.

RASKAR, R., TAN, K.-H., FERIS, R., YU, J., AND TURK, M. 2004. Non-photorealistic camera: depth edge detection and stylized rendering using multi-flash imaging. *ACM Trans. Graph. 23*, 3, 679–688.

ROUSSOU, M., AND DRETTAKIS, G. 2003. Photorealism and non-photorealism in virtual heritage representation. In *VAST 2003 and First Eurographics Workshop on Graphics and Cultural Heritage*, A.Chalmers, D.Arnold, and F. Niccolucci, Eds., Eurographics.

SAITO, T., AND TAKAHASHI, T. 1990. Comprehensible rendering of 3-d shapes. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, ACM Press, 197–206.

SALISBURY, M. P., ANDERSON, S. E., BARZEL, R., AND SALESIN, D. H. 1994. Interactive pen-and-ink illustration. In *Proceedings of SIGGRAPH 94*, 101–108.

STROTHOTTE, T., AND SCHLECHTWEG, S. 2002. *Non-Photorealistic Computer Graphics: Modeling, Rendering and Animation*, 1 ed. Morgan Kaufmann, June 15. ISBN: 1558607870, pages 472.

SUGANO, N., KATO, H., AND TACHIBANA, K. 2003. The effects of shadow representation of virtual objects in augmented reality. In *ISMAR*, 76–83.

WYNN, C. 2002. OpenGL Render-to-Texture. TR, nVIDIA Corporation, March.